CENSUS

IT Security Works

# vs com.apple.security.sandbox


PATROKLOS ARGYROUDIS
**CENSUS S.A.**

argp@census-labs.com
**www.census-labs.com**

# Who am I


- Computer security researcher at CENSUS S.A.
  - Vulnerability research, RE, exploit development
- Before CENSUS: PhD and Postdoc at TCD doing netsec
- Heap exploitation obsession (userland & kernel)
- Wrote some Phrack papers

# Introduction


- This talk is on reverse engineering the iOS com.apple.security.sandbox kernel extension (aka sandbox.kext)

- iOS-specific unless otherwise noted

- Tested on up to latest stable iOS: 12.1.4 (build 16D57)
  - And latest beta: 12.2 beta 5 (build 16E5223a)

# Outline


- Sandbox overview
- iOS sandbox implementation details
- Sandbox.kext reversing engineering
- Results (findings, attack surface, sandbox escape notes)

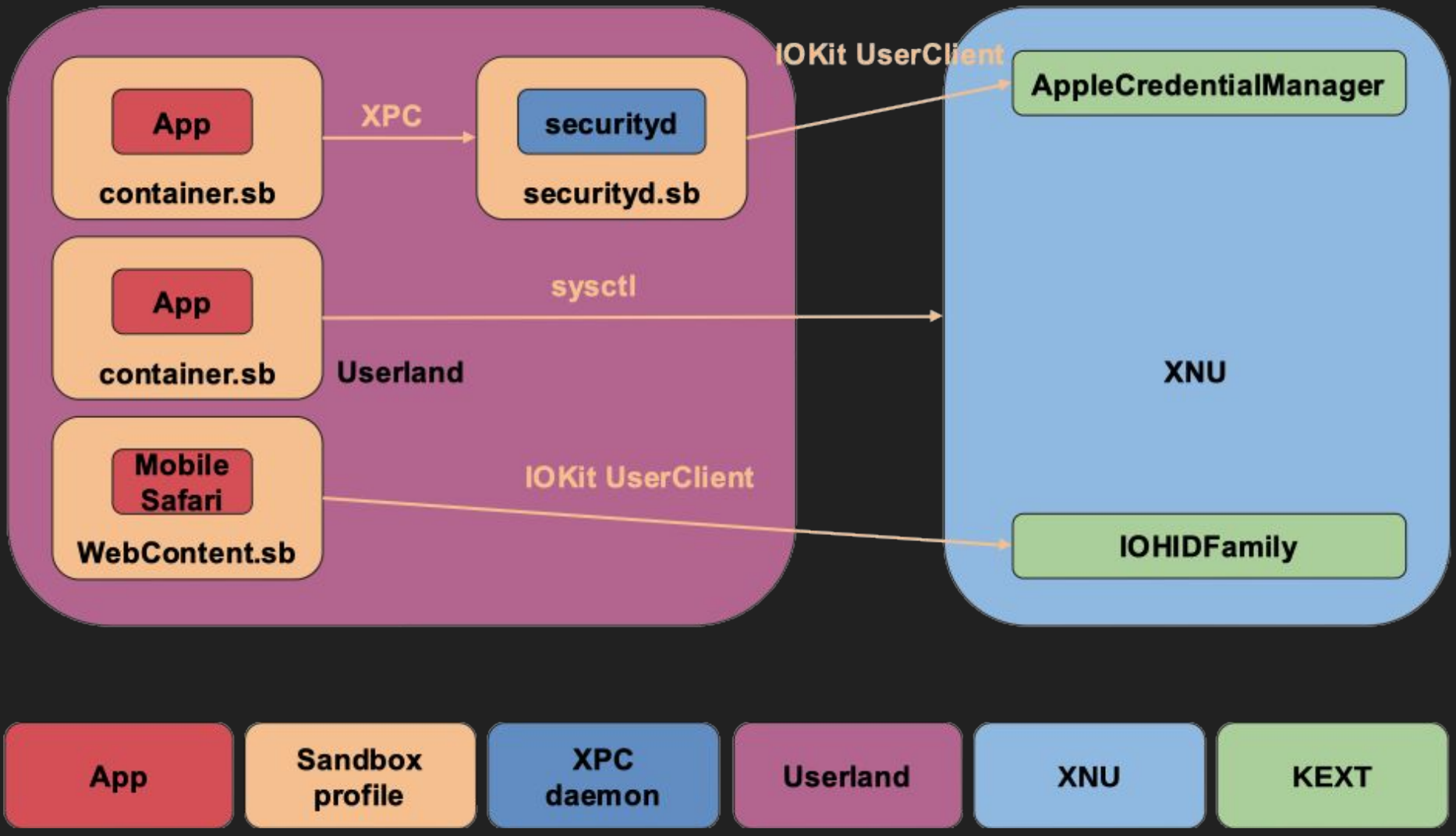
# Sandbox overview


- A sandbox is a technology that protects an underlying system, by limiting the operations an app that runs on the system can perform
  - White-list, obviously, so sandbox technologies specify what is allowed (policies or profiles)

- On iOS the sandbox protects mainly two resources of the underlying system
  - The kernel and its drivers (kernel extensions)
  - IPC (XPC, NSXPC, etc.) system services
  - Others too (e.g. parts of the filesystem)

# Sandbox overview

CENSUS
IT Security Works



# Sandbox vs privilege escalation


- Apple introduced the sandbox in iOS 2.0 and relies on it a lot for limiting privilege escalation and post-exploitation
  - Every iOS release keeps reducing the surface accessible from within the sandbox
  - Sandbox escape, then kernel interface to LPE

- Apps are placed in a container (app sandbox) by default
  - They don't need to do anything code-wise (or in any other way)
  - AMFI (Apple Mobile File Integrity) for code signing and entitlements (not discussed - out of scope)

# iOS sandbox implementation


- Based on the TrustedBSD (FreeBSD) MACF (Mandatory Access Control Framework)
  - MAC: security policy is centrally controlled by a security policy administrator; Apple in our case
  - Users do not have the ability to override the policy and, for example, grant access to files that would otherwise be restricted
  - Not even ones they own/created

- Enforced by the kernel

# Example: IOKit properties


- IOKit drivers allow the getting/setting of their properties from userland
  - Userland API leads to kernel function is_io_registry_entry_get_property

- MACF specific code addition (#ifdef CONFIG_MACF)

```
2995
2996  /* Routine io_registry_entry_get_property */
2997  kern_return_t is_io_registry_entry_get_property(
2998          io_object_t registry_entry,
2999          io_name_t property_name,
3000          io_buf_ptr_t *properties,
3001          mach_msg_type_number_t *propertiesCnt )
3002  {
3003      kern_return_t       err;
3004      vm_size_t           len;
3005      OSObject *          obj;
3006
3007      CHECK( IORegistryEntry, registry_entry, entry );
3008
3009  #if CONFIG_MACF
3010      if (0 != mac_iokit_check_get_property(kauth_cred_get(), entry, property_name))
3011          return kIOReturnNotPermitted;
3012  #endif
```

# mac_iokit_check_get_property


- One example of an entry function to MACF for making an access control decision (IOKit get property here)

```
104  int
105  mac_iokit_check_get_property(kauth_cred_t cred, io_object_t registry_entry, const char *name)
106  {
107          int error;
108
109          MAC_CHECK(iokit_check_get_property, cred, registry_entry, name);
110          return (error);
111  }
```

```
254  #define MAC_CHECK(check, args...) do {
255          struct mac_policy_conf *mpc;
256          u_int i;
257
258          error = 0;
259          for (i = 0; i < mac_policy_list.staticmax; i++) {
260                  mpc = mac_policy_list.entries[i].mpc;
261                  if (mpc == NULL)
262                          continue;
263
264                  if (mpc->mpc_ops->mpo_ ## check != NULL)
265                          error = mac_error_select(
266                              mpc->mpc_ops->mpo_ ## check (args),
267                              error);
```

**callback**

**caller's credentials**

**driver**

**property name**

# mpo_iokit_check_get_property


- Actual implementation of the check; policy hook or operation
  - MAC_CHECK macro checks the operation against policy modules; the sandbox is one of them (the other is AMFI)
  - Struct that holds all policy hooks (operations)
  - Not the same on macOS and iOS (XNU, kexts)

- const static struct mac_policy_ops policy_ops (macOS):

```
510             CHECK_SET_HOOK(proc_check_proc_info)
511
512             CHECK_SET_HOOK(vnode_notify_link)
513
514             CHECK_SET_HOOK(iokit_check_filter_properties)
515             CHECK_SET_HOOK(iokit_check_get_property)
516     };
```

# Sandbox profiles


- Each hook implements a check (implemented in the kernel)
  - Specifically called at certain code points as we saw
  - These hooks/operations are used in profiles
  - Profiles specify allowed operations and conditions on them

- During the kernel's (or a kext's) initialization mac_policy_register is called
  - Registers hooks (operations) from the mac_policy_ops struct
  - Calls hook_policy_init which loads the sandbox profiles

# iOS sandbox implementation


- Closed source both on iOS and on macOS
  - Sandbox.kext binary on macOS has symbols
  - Policies (profiles) on macOS' filesystem: /System/Library/Sandbox/Profiles
  - Sandbox Profile Language (SBPL) - (Tiny)Scheme
  - No container.sb there

- On iOS profiles compiled and packed in the kext itself
  - Sandbox.kext binary has no symbols
  - Only some strings (that can aid symbolization/RE)

# Operations


- An operation is some action that an app wants to perform that is checked by the sandbox
  - Abstract names (labels) corresponding to MACF callbacks
  - Callbacks defined in security/mac_policy.h
  - Implemented in the sandbox kext

```
[argp@mole ~/projects/ios/12/sandbox_iX_12_1_b4_16B5084a_profiles]$ grep get-properties * | head -n 10
accessoryd.sb:693:(allow iokit-get-properties)
AdSheet.sb:1756:(allow iokit-get-properties)
adtrackingd.sb:637:(allow iokit-get-properties
afcd.sb:1032:(allow iokit-get-properties)
AGXCompilerService.sb:655:(allow iokit-get-pro
```

**iokit-get-properties label / operation**

```
1419  );
1420  /**
1421    @brief Access control check for getting I/O Kit device properties
1422    @param cred Subject credential
1423    @param entry Target device
1424    @param name Property name
1425
1426    Determine whether the subject identified by the credential can get
1427    properties on an I/O Kit device.
1428
1429    @return Return 0 if access is granted, or an appropriate value for
1430    errno.
1431  */
1432  typedef int mpo_iokit_check_get_property_t(
1433          kauth_cred_t cred,
1434          io_object_t entry,
1435          const char *name
1436  );
```

# CONFIG_MACF callbacks


```
2995
2996  /* Routine io_registry_entry_get_property */
2997  kern_return_t is_io_registry_entry_get_property(
2998          io_object_t registry_entry,
2999          io_name_t property_name,
3000          io_buf_ptr_t *properties,
3001          mach_msg_type_number_t *propertiesCnt )
3002  {
3003      kern_return_t       err;
3004      vm_size_t           len;
3005      OSObject *          obj;
3006
3007      CHECK( IORegistryEntry, registry_entry, entry );
3008
3009  #if CONFIG_MACF
3010      if (0 != mac_iokit_check_get_property(kauth_cred_get(), entry, property_name))
3011          return kIOReturnNotPermitted;
3012  #endif
```

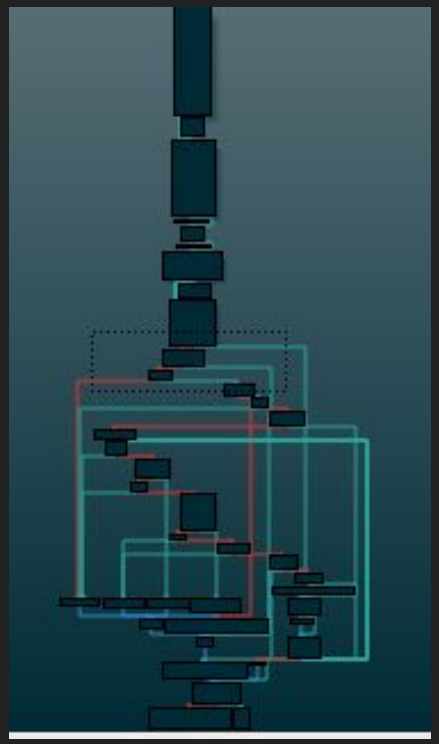
```
104  int
105  mac_iokit_check_get_property(kauth_cred_t cred, io_object_t registry_entry, const char *name)
106  {
107          int error;
108
109          MAC_CHECK(iokit_check_get_property, cred, registry_entry, name);
110          return (error);
111  }
```

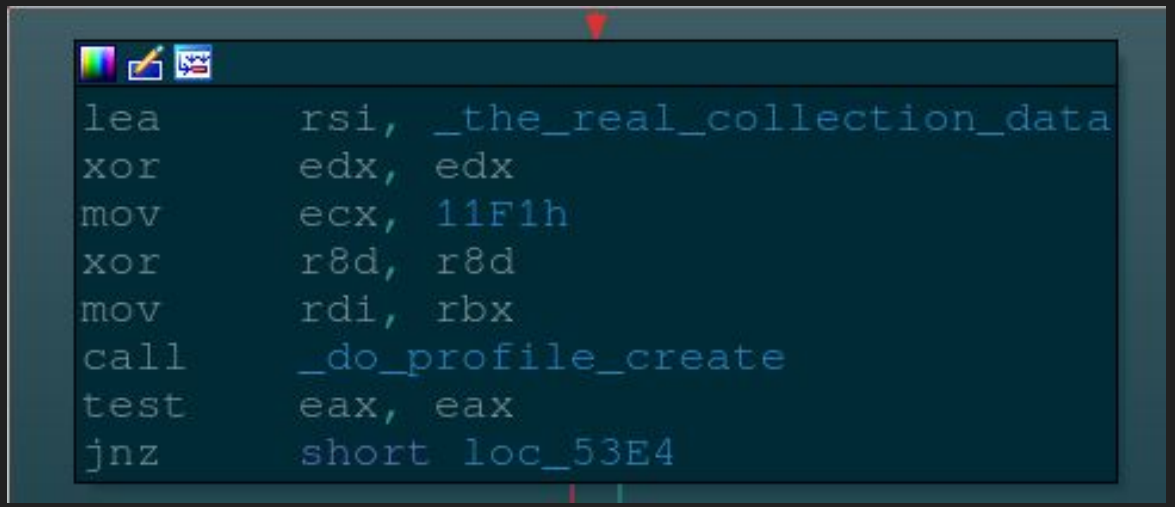
```
254  #define MAC_CHECK(check, args...) do {
255          struct mac_policy_conf *mpc;
256          u_int i;
257
258          error = 0;
259          for (i = 0; i < mac_policy_list.staticmax; i++) {
260                  mpc = mac_policy_list.entries[i].mpc;
261                  if (mpc == NULL)
262                          continue;
263
264                  if (mpc->mpc_ops->mpo_ ## check != NULL)
265                          error = mac_error_select(
266                              mpc->mpc_ops->mpo_ ## check (args),
267                              error);
```

# Policies (profiles) -- macOS


- MACF callbacks check against loaded profiles
  - How are they loaded?

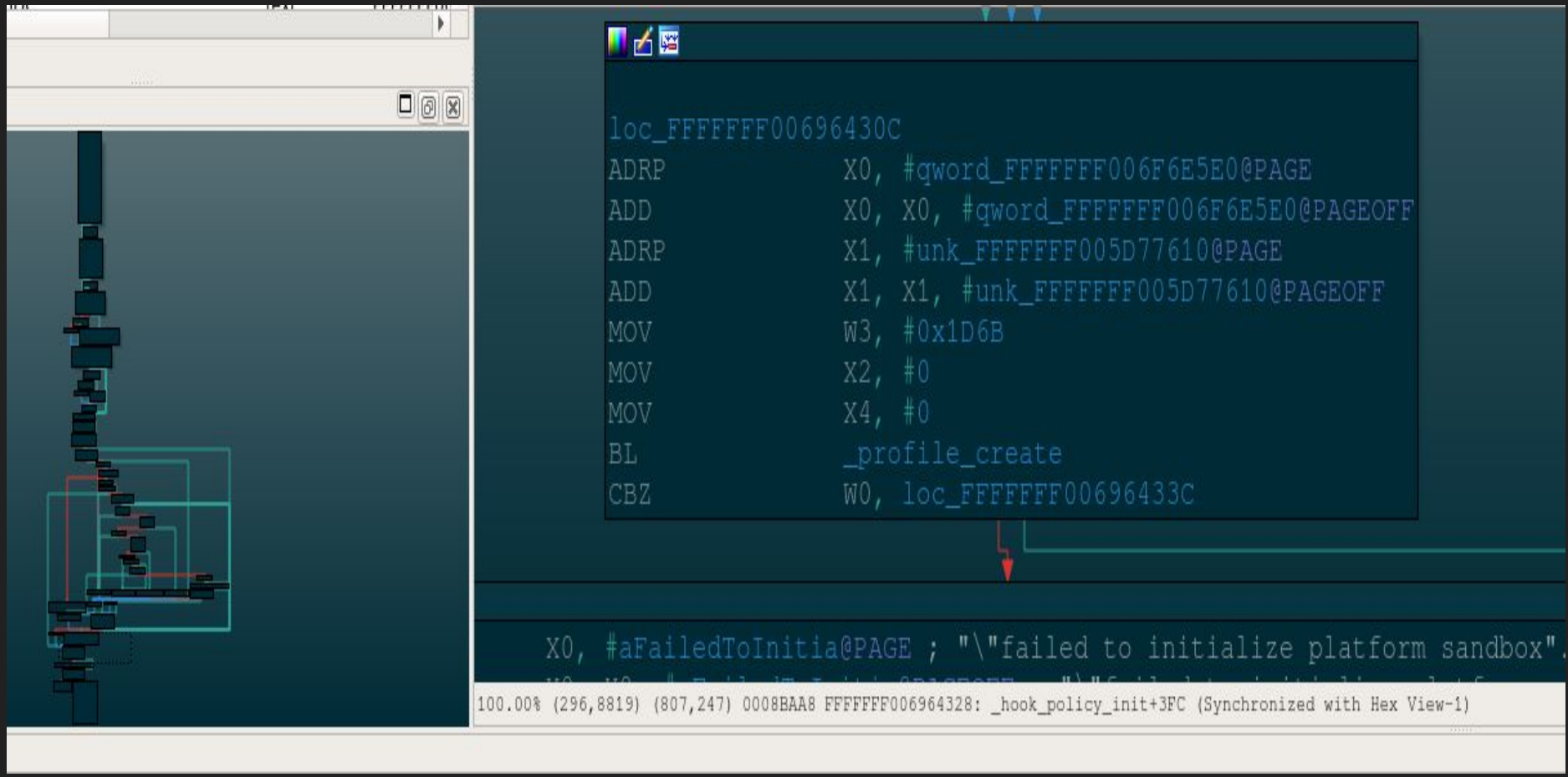
```
44  static void
45  hook_policy_init(struct mac_policy_conf *mpc)
46  {
47          printf("Policy '%s' = '%s' ready\n", mpc->mpc_name, mpc->mpc_fullname);
48  }
```





# Policies (profiles) -- iOS

CENSUS
IT Security Works



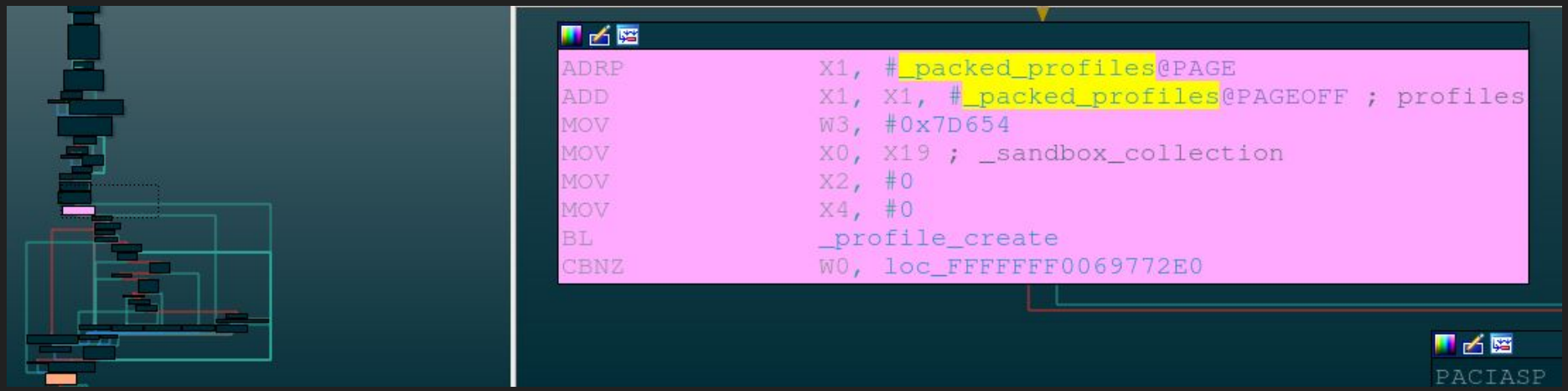
# _profile_create


- X0: pointer to heap (sandbox profile context buffer)
- X1: __const address (kext Mach-O) with packed sandbox data
- X2: flag
- X3: size of the packed data at X1
- X4: flag

- How to find _profile_create?
  - String “re_cache”
  - Called twice with two different __const addresses
  - (Called in _hook_policy_init, also useful to have)
  - Both of these addresses useful for more RE

# _profile_create (profiles)

CENSUS
IT Security Works



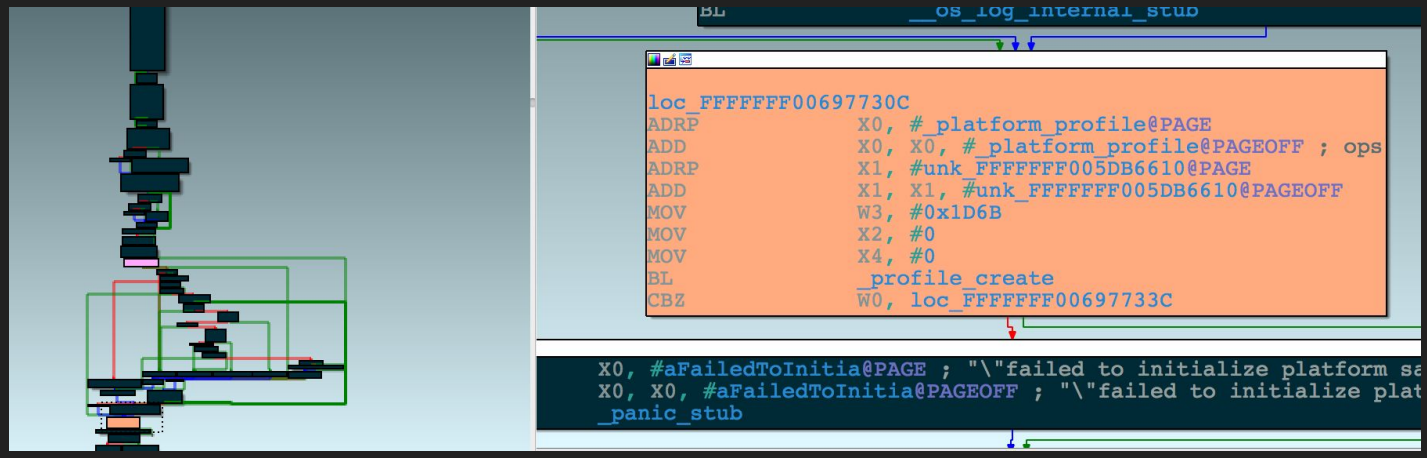
```
ADRP            X1, #_packed_profiles@PAGE
ADD             X1, X1, #_packed_profiles@PAGEOFF ; profiles
MOV             W3, #0x7D654
MOV             X0, X19 ; _sandbox_collection
MOV             X2, #0
MOV             X4, #0
BL              _profile_create
CBNZ            W0, loc_FFFFFFF0069772E0
```

PACIASP

```
__const:FFFFFFF005DB8380 _packed_profiles DCB     0          ; DATA XREF: sub_FFFFFFF0
__const:FFFFFFF005DB8380                                     ; sub_FFFFFFF006975B14+34
__const:FFFFFFF005DB8381                  DCB 0x80
__const:FFFFFFF005DB8382                  DCB 0x8B
__const:FFFFFFF005DB8383                  DCB 0xAD
__const:FFFFFFF005DB8384                  DCB 0xBA
__const:FFFFFFF005DB8385                  DCB 0xAD
__const:FFFFFFF005DB8386                  DCB 0xBB
__const:FFFFFFF005DB8387                  DCB 0xAD
__const:FFFFFFF005DB8388                  DCB 0xBA
__const:FFFFFFF005DB8389                  DCB     0
__const:FFFFFFF005DB838A                  DCB     3
__const:FFFFFFF005DB838B                  DCB     3
__const:FFFFFFF005DB838C                  DCB 0xC1
__const:FFFFFFF005DB838D                  DCB     0
__const:FFFFFFF005DB838E                  DCB 0xBC
__const:FFFFFFF005DB838F                  DCB 0xAD
__const:FFFFFFF005DB8390                  DCB     0
__const:FFFFFFF005DB8391                  DCB     0
__const:FFFFFFF005DB8392                  DCB 0x8A
__const:FFFFFFF005DB8393                  DCB 0xAD
__const:FFFFFFF005DB8394                  DCB 0x8A
__const:FFFFFFF005DB8395                  DCB 0xAD
__const:FFFFFFF005DB8396                  DCB 0x8A
__const:FFFFFFF005DB8397                  DCB 0xAD
__const:FFFFFFF005DB8398                  DCB 0x8A
__const:FFFFFFF005DB8399                  DCB 0xAD
```

# _profile_create (operations)

CENSUS
IT Security Works

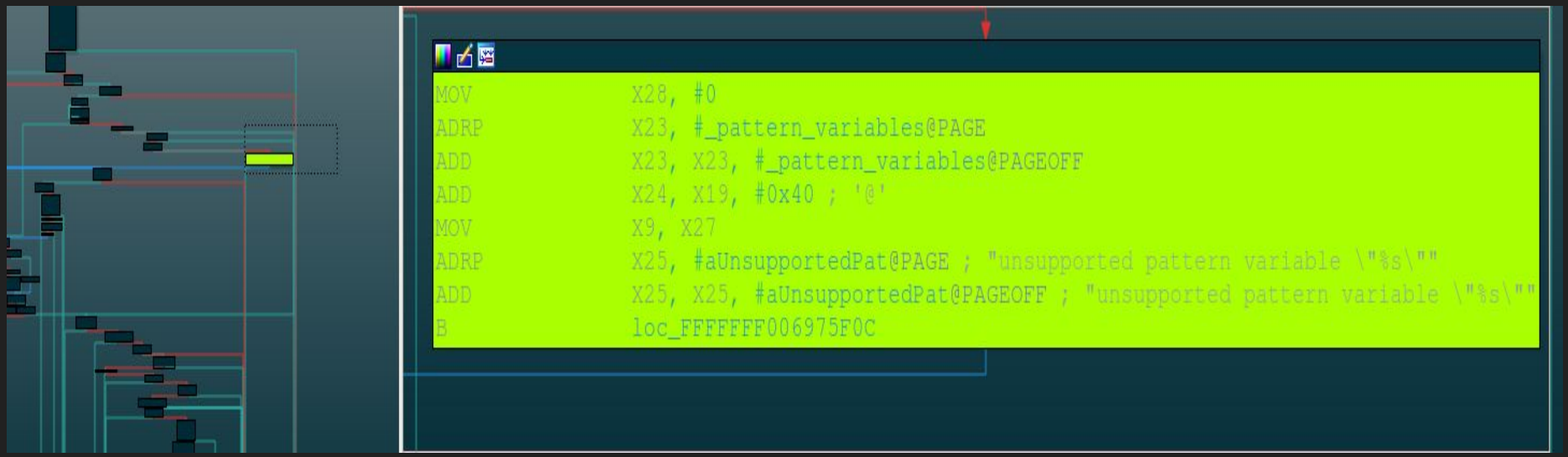



```
__const:FFFFFFF006F6DA60 _platform_profile DCQ 0                    ; DATA XREF: _hook_policy_ini
__const:FFFFFFF006F6DA60                                            ; _hook_policy_init+3E4↑o ...
__const:FFFFFFF006F6DA68 off_FFFFFFF006F6DA68 DCQ aDefault          ; DATA XREF: sub_FFFFFFF00697
__const:FFFFFFF006F6DA68                                            ; sub_FFFFFFF00697D374+74C↑o
__const:FFFFFFF006F6DA68                                            ; "default"
__const:FFFFFFF006F6DA70                DCQ aAppleeventSend         ; "appleevent-send"
__const:FFFFFFF006F6DA78                DCQ aAuthorizationR         ; "authorization-right-obtain
__const:FFFFFFF006F6DA80                DCQ aBootArgSet             ; "boot-arg-set"
__const:FFFFFFF006F6DA88                DCQ aDevice                 ; "device*"
__const:FFFFFFF006F6DA90                DCQ aDeviceCamera           ; "device-camera"
__const:FFFFFFF006F6DA98                DCQ aDeviceMicropho         ; "device-microphone"
__const:FFFFFFF006F6DAA0                DCQ aDarwinNotifica         ; "darwin-notification-post"
__const:FFFFFFF006F6DAA8                DCQ aDistributedNot         ; "distributed-notification-p
__const:FFFFFFF006F6DAB0                DCQ aDynamicCodeGen         ; "dynamic-code-generation"
__const:FFFFFFF006F6DAB8                DCQ aFile                   ; "file*"
__const:FFFFFFF006F6DAC0                DCQ aFileChroot             ; "file-chroot"
__const:FFFFFFF006F6DAC8                DCQ aFileClone              ; "file-clone"
__const:FFFFFFF006F6DAD0                DCQ aFileIoctl              ; "file-ioctl"
__const:FFFFFFF006F6DAD8                DCQ aFileIssueExten         ; "file-issue-extension"
__const:FFFFFFF006F6DAE0                DCQ aFileLink               ; "file-link"
__const:FFFFFFF006F6DAE8                DCQ aFileMapExecuta         ; "file-map-executable"
__const:FFFFFFF006F6DAF0                DCQ aFileMknod              ; "file-mknod"
__const:FFFFFFF006F6DAF8                DCQ aFileMount              ; "file-mount"
__const:FFFFFFF006F6DB00                DCQ aFileMountUpdat         ; "file-mount-update"
__const:FFFFFFF006F6DB08                DCQ aFileRead               ; "file-read*"
__const:FFFFFFF006F6DB10                DCQ aFileReadData           ; "file-read-data"
__const:FFFFFFF006F6DB18                DCQ aFileReadMetada         ; "file-read-metadata"
__const:FFFFFFF006F6DB20                DCQ aFileReadXattr          ; "file-read-xattr"
__const:FFFFFFF006F6DB28                DCQ aFileRevoke             ; "file-revoke"
__const:FFFFFFF006F6DB30                DCQ aFileSearch             ; "file-search"
```

# In _profile_create (pattern variables)

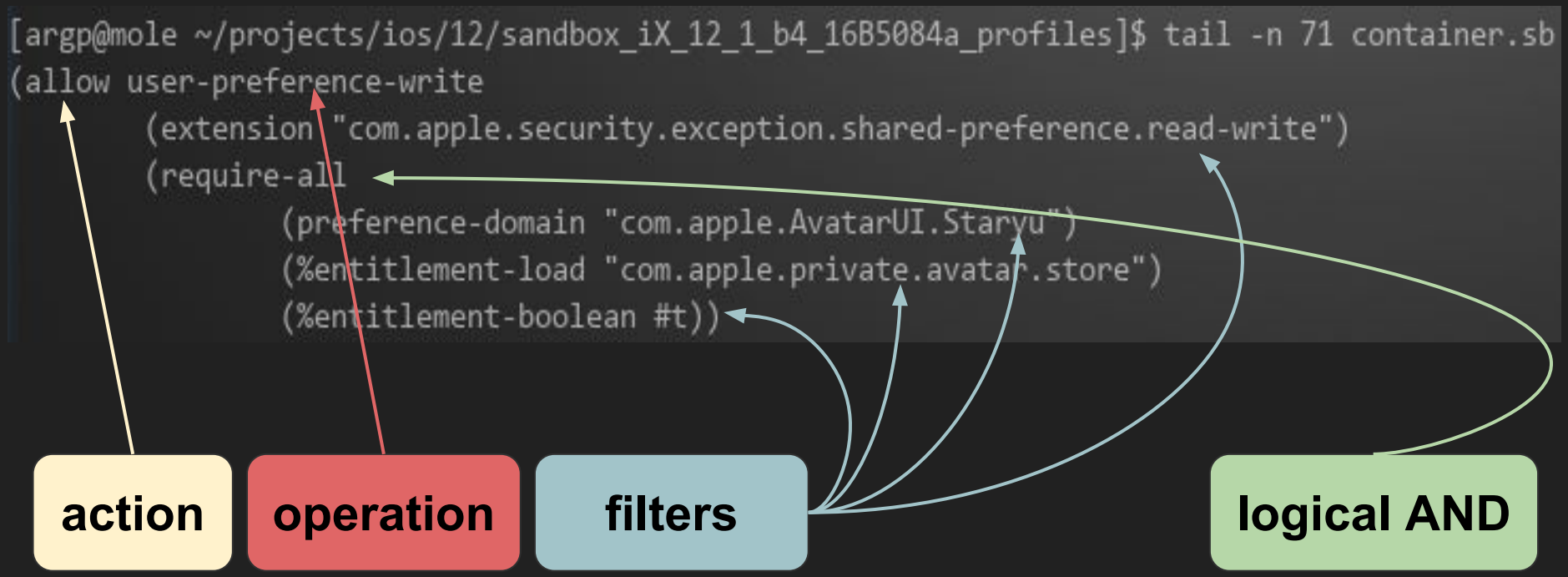
CENSUS
IT Security Works

```
[argp@mole ~/projects/ios/12/sandbox_iX_12_1_b4_16B5084a_profiles]$ grep HOME sharingd.sb | head -n 10
105:                            (subpath-prefix "${HOME}/Downloads/com.apple.AirDrop"))
116:                                            (subpath-prefix "${HOME}/Library/Mobile Documents")
124:                                            (subpath-prefix "${HOME}/Media"))))
129:                                            (subpath-prefix "${HOME}/Library/Mobile Documents")
```




```
DF08 _pattern_variables DCQ aHome          ; DATA XREF: _profile_create+180↑o
DF08                                       ; _profile_create+184↑o
DF08                                       ; "HOME"
DF10                    DCQ aFrontUserHome ; "FRONT_USER_HOME"
DF18                    DCQ aProcessTempDir ; "PROCESS_TEMP_DIR"
```

# SBPL notes




- Action or “decision”
- Logical OR == “require-any”
- All together == rule

# /usr/lib/libsandbox.1.dylib


- Sandox Policy Language (SBPL -- TinyScheme) compiler
  - Exposes an API
  - Also the dylib is symbolized

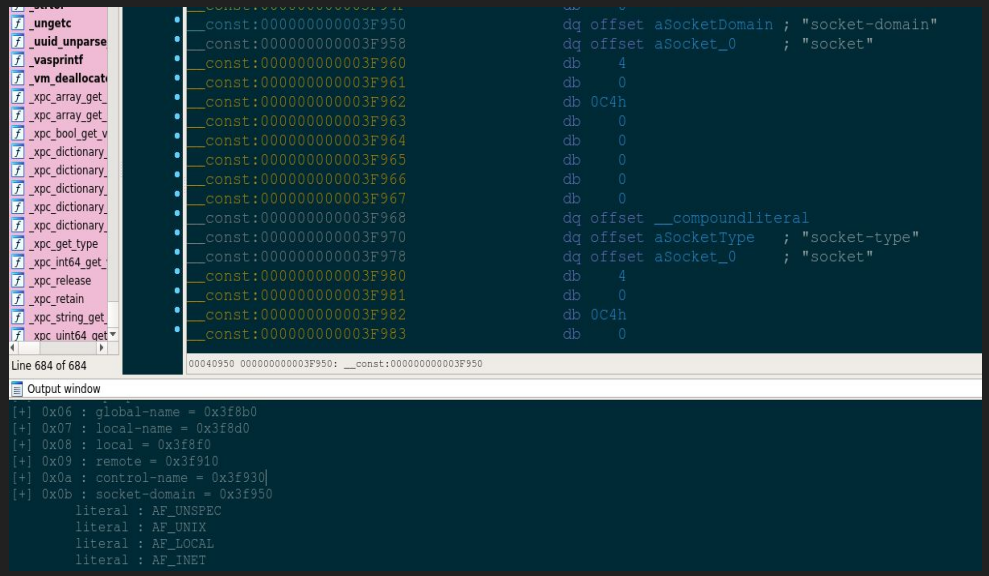
- All filters and their literals!
  - _filter_info
  - find_filters.py demo

- Verification with a ctypes script that uses the API
  - libpysandbox_compile.py demo

# find_filters.py demo

CENSUS
IT Security Works

```python
import idc
import idautils
import idaapi
import ida_bytes

true = True
false = False
none = None

def main():
    filter_info_ea = idc.LocByName("_filter_info")
    print "[+] filter_info_ea : 0x%x\n" % (filter_info_ea)

    ea = filter_info_ea + 0x20
    i = 0x1

    f = open("/tmp/find_filters_out.json", "w")
    f.write("{\n")

    while true:
        filter_str = idc.get_strlit_contents(Qword(ea))

        if filter_str == "ip":
            break

        print "[+] 0x%x : %s : 0x%x" % (i, filter_str, ea)

        f.write("  \"0x%x\" :\n" % (i))
        f.write("  {\n")
        f.write("    \"name\" : \"%s\",\n" % (filter_str))

        literals_ea = Dword(ea + 0x18)

        if literals_ea != 0:
            if Name(literals_ea).startswith("__compoundliteral"):
                # print "  literals_ea : %s : 0x%x" % (Name(literals_ea), literals_ea)

                lea = literals_ea
                j = 0

                while true:
                    lea_deref = Dword(lea)

                    if lea_deref == 0x0:
                        break

                    literal = idc.get_strlit_contents(lea_deref)
                    # print "  literal : %s" % (literal)

                    f.write("    \"literal_%d\" : \"%s\",\n" % (j, literal))

                    lea = lea + 0x10
                    j = j + 1
```




```
_data:0000000000046030 __compoundliteral dq offset aAfUnspec   ; DATA XREF: __co
_data:0000000000046030                                         ; "AF_UNSPEC"
_data:0000000000046038                 align 20h
_data:0000000000046040                 dq offset aAfUnix       ; "AF_UNIX"
_data:0000000000046048                 db    1
_data:0000000000046049                 db    0
_data:000000000004604A                 db    0
_data:000000000004604B                 db    0
_data:000000000004604C                 db    0
_data:000000000004604D                 db    0
_data:000000000004604E                 db    0
_data:000000000004604F                 db    0
_data:0000000000046050                 dq offset aAfLocal      ; "AF_LOCAL"
```

# libpysandbox_compile.py demo


```
1   import json
2   import ctypes
3   import binascii
4   import difflib
5
6   sb_scheme = """
7   (version 1)
8
9   ;;; Allow registration of per-pid services.
10  (allow mach-register (local-name-prefix ""))
11
12  ;;; Allow lookup of XPC services for backward-compatibility.
13  (allow mach-lookup (xpc-service-name-prefix ""))
14
15  ;;; Allow system processes to trigger auto-mounting of filesystems.
16  (allow system-automount
17          (process-attribute is-sandboxed))
18
19  (allow system-socket
20           (socket-domain AF_ROUTE))
21
22  (allow mach-lookup
23          (global-name "com.apple.appsleep")
24          (global-name "com.apple.bsd.dirhelper")
25          (global-name "com.apple.cfprefsd.agent")
26          (global-name "com.apple.cfprefsd.daemon")
27          (global-name "com.apple.diagnosticd")
28          (global-name "com.apple.dyld.closured")
29          (global-name "com.apple.espd")
30          (global-name "com.apple.logd")
31          (global-name "com.apple.logd.events")
32          (global-name "com.apple.secinitd")
33          (global-name "com.apple.system.DirectoryService.libinfo_v1")
34          (global-name "com.apple.system.logger")
35          (global-name "com.apple.system.notification_center")
36          (global-name "com.apple.system.opendirectoryd.libinfo")
37          (global-name "com.apple.system.opendirectoryd.membership")
38          (global-name "com.apple.trustd")
39          (global-name "com.apple.trustd.agent")
40          (global-name "com.apple.xpc.activity.unmanaged")
41          (global-name "com.apple.xpcd")
42          (local-name "com.apple.cfprefsd.agent"))
43  """
```

# libpysandbox_compile.py demo


```
60      sb.sandbox_compile_string.restype = ctypes.POINTER(sandbox_profile)
61      profile_obj = sb.sandbox_compile_string(sb_scheme)[0]
62
63      sb_bin = (ctypes.c_char * profile_obj.len).from_address(profile_obj.content)
64      sb_hex = binascii.hexlify(sb_bin)
65
66      print "socket-domain AF_ROUTE:"
67      print sb_hex
68
69      sb_scheme2 = sb_scheme.replace("AF_ROUTE", "AF_UNIX")
70
71      f = open("/tmp/tmp.sb", "w")
72      f.write(sb_scheme2)
73      f.close()
74
75      sb.sandbox_compile_file.restype = ctypes.POINTER(sandbox_profile)
76      sb.sandbox_compile_file.argtypes = [ctypes.c_char_p, ctypes.c_uint, ctypes.c_uint]
77      profile_obj = sb.sandbox_compile_file("/tmp/tmp.sb", 0, error)[0]
78
79      sb_bin = (ctypes.c_char * profile_obj.len).from_address(profile_obj.content)
80      sb_hex2 = binascii.hexlify(sb_bin)
81
82      print "\nsocket-domain AF_UNIX:"
83      print sb_hex2
84
85      diff = difflib.ndiff(sb_hex, sb_hex2)
```

# Filters


- Filters use literals (we found them)
  - and/or can use regular expressions
  - Tried to reverse regexes based on Esser's and Dion's work
  - Then found Sandblaster, used their regular expressions deserialization work (used it as a module in my IDAPython script)

- You can think of filters as conditions applying on operations
  - Packed in the "ops_filters_struct"

# Packed profiles' structure


- Header
  - iOS version magic
  - Regexes offsets array offset
  - Regexes count
  - Profiles count
- ops_filters_struct

```
_const:FFFFFFF005DB8380 _packed_profiles DCB    0          ; DATA XREF: sub_FFFFFFF006975B1
_const:FFFFFFF005DB8380                                    ; sub_FFFFFFF006975B14+34↓o ...
_const:FFFFFFF005DB8381                 DCB 0x80           ; iOS version magic
_const:FFFFFFF005DB8382                 DCB 0x8B           ; regexes offsets struct offset
_const:FFFFFFF005DB8383                 DCB 0xAD
_const:FFFFFFF005DB8384                 DCB 0xBA
_const:FFFFFFF005DB8385                 DCB 0xAD
_const:FFFFFFF005DB8386                 DCB 0xBB
_const:FFFFFFF005DB8387                 DCB 0xAD
_const:FFFFFFF005DB8388                 DCB 0xBA           ; regexes count == 0xBA
_const:FFFFFFF005DB8389                 DCB    0
_const:FFFFFFF005DB838A                 DCB    3
_const:FFFFFFF005DB838B                 DCB    3
_const:FFFFFFF005DB838C                 DCB 0xC1           ;  profiles count == 0xC1
```

```
__const:FFFFFFF005DC65C0                DCB    0                ; ops_filters_struct
__const:FFFFFFF005DC65C1                DCB    6
__const:FFFFFFF005DC65C2                DCB 0xE3
__const:FFFFFFF005DC65C3                DCB 0xB0
__const:FFFFFFF005DC65C4                DCB 0x89
__const:FFFFFFF005DC65C5                DCB 0xAD
__const:FFFFFFF005DC65C6                DCB 0x49 ; I
__const:FFFFFFF005DC65C7                DCB 0x1C
__const:FFFFFFF005DC65C8                DCB    0
__const:FFFFFFF005DC65C9                DCB    6
__const:FFFFFFF005DC65CA                DCB 0x43 ; C
__const:FFFFFFF005DC65CB                DCB 0xB1
__const:FFFFFFF005DC65CC                DCB 0x89
__const:FFFFFFF005DC65CD                DCB 0xAD
__const:FFFFFFF005DC65CE                DCB 0x4A ; J
__const:FFFFFFF005DC65CF                DCB 0x1C
```

# Packed profiles' structure


- Regexes offsets array

```
__const:FFFFFFF005E0EFD8    DCB 0x66 ; f        ; regexes offsets struct
__const:FFFFFFF005E0EFD9    DCB 0xAE
__const:FFFFFFF005E0EFDA    DCB 0x53 ; S
__const:FFFFFFF005E0EFDB    DCB 0xAE
__const:FFFFFFF005E0EFDC    DCB 0x42 ; B
__const:FFFFFFF005E0EFDD    DCB 0xAE
__const:FFFFFFF005E0EFDE    DCB 0xB3
__const:FFFFFFF005E0EFDF    DCB 0xAE
__const:FFFFFFF005E0EFE0    DCB 0xA6
__const:FFFFFFF005E0EFE1    DCB 0xAE
__const:FFFFFFF005E0EFE2    DCB 0x97
__const:FFFFFFF005E0EFE3    DCB 0xAE
__const:FFFFFFF005E0EFE4    DCB    1
__const:FFFFFFF005E0EFE5    DCB 0xAE
```

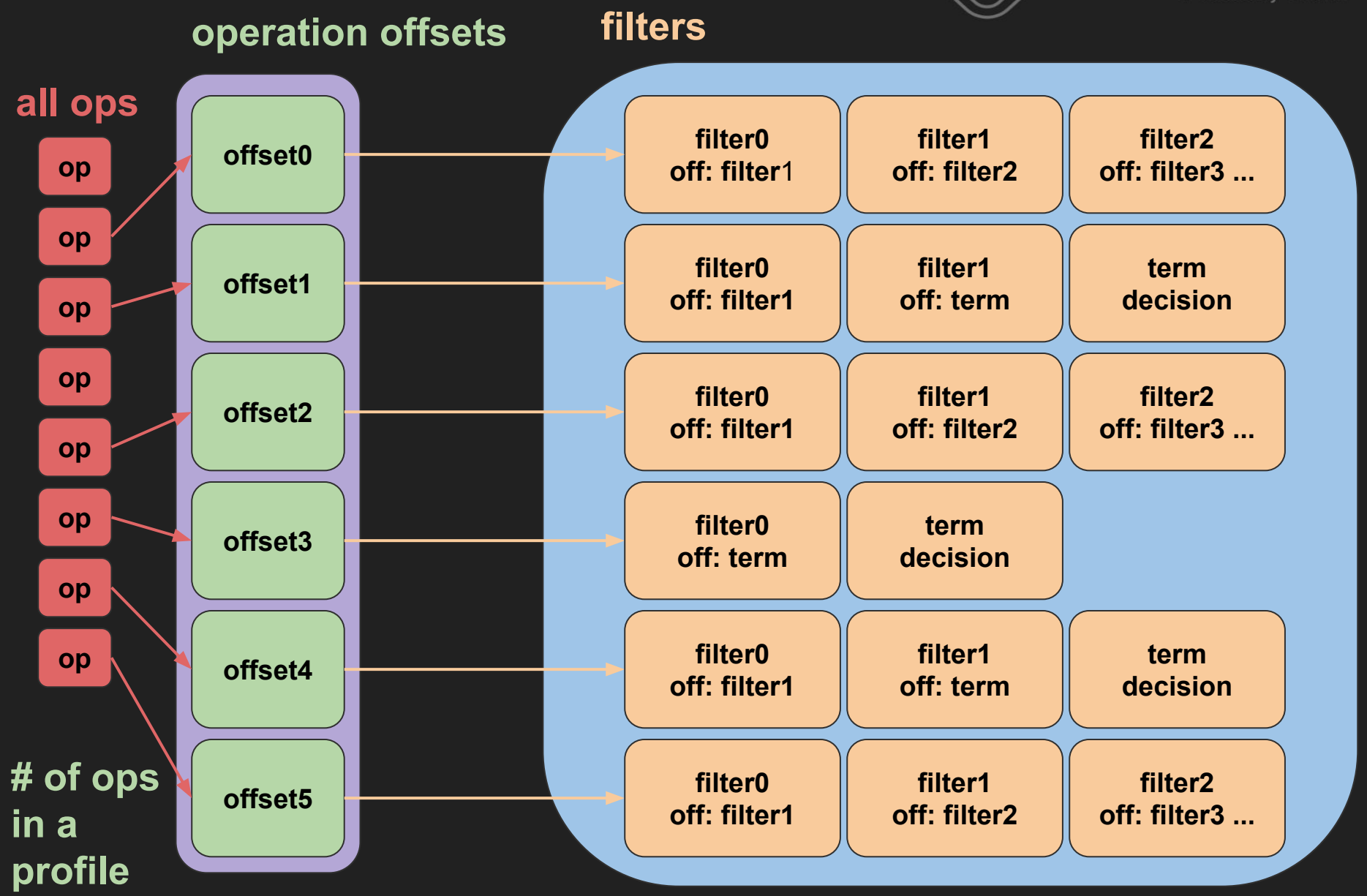
- Regexes
  - Regex len
  - Regex bytes

```
__const:FFFFFFF005E0F6B8    DCB 0x6B ; k        ; regexes
__const:FFFFFFF005E0F6B9    DCB    0
__const:FFFFFFF005E0F6BA    DCB 0x19
__const:FFFFFFF005E0F6BB    DCB    2
__const:FFFFFFF005E0F6BC    DCB 0x2F ; /
__const:FFFFFFF005E0F6BD    DCB    2
__const:FFFFFFF005E0F6BE    DCB 0x70 ; p
__const:FFFFFFF005E0F6BF    DCB    2
__const:FFFFFFF005E0F6C0    DCB 0x72 ; r
__const:FFFFFFF005E0F6C1    DCB    2
__const:FFFFFFF005E0F6C2    DCB 0x69 ; i
```

# Regexes parsing


```
1   regexes = {}
2
3   header = ida_bytes.get_word(x1_addr)
4   print "[xxx] header = 0x%x" % (header)
5
6   regexes_offsets_array_offset_ea = x1_addr + 2
7   regexes_offsets_array_offset = ida_bytes.get_word(regexes_offsets_array_offset_ea)
8
9   print "[xxx] regexes_offsets_array_offset at: 0x%x" % (regexes_offsets_array_offset_ea)
10  print "[xxx] regexes_offsets_array_offset = 0x%x" % (regexes_offsets_array_offset)
11
12  regexes_offsets = []
13
14  print "[xxx] regexes_offsets at: 0x%x" % (x1_addr + (regexes_offsets_array_offset * 8))
15
16  for i in range(0, regexes_count):
17      offset = ida_bytes.get_word(x1_addr + (regexes_offsets_array_offset * 8) + (i * 2))
18      regexes_offsets.append(offset)
19
20  for offset in regexes_offsets:
21      regex_len = ida_bytes.get_32bit(x1_addr + (offset * 8))
22
23      regex_bytes = []
24
25      for i in range(0, regex_len):
26          re_byte = ida_bytes.get_byte(x1_addr + (offset * 8) + 4 + i)
27          re.append(re_byte)
28
29      regexes[offset] = regex_bytes
```

# ops_filters_struct

# Filters


- Filters may use regexes and/or literals
- Indexes to regexes_offset_array
  - Offset to actual regex bytes
- Same for literals (strings)

# ops_filters_struct parsing


```
    for i in range(0, sb_profiles_count):
        ops_filters_struct = []

        profile_name_offset_ea = x1_addr + 14 + (len(operation_names) + 2) * 2 * i

        profile_name_offset = ida_bytes.get_word(profile_name_offset_ea)
        profile_name = unpack_str(x1_addr, profile_name_offset)

        print "[+] Processing sandbox profile: %s" % (profile_name)

        for j in range(0, len(operation_names)):
            op_offset_ea = x1_addr + 14 + (len(operation_names) + 2) * 2 * i + 4 + j * 2

            op_offset = ida_bytes.get_word(op_offset_ea)

            ops_filters_struct.insert(j, op_offset)


def unpack_str(packed_profiles_ea, offset):
    str_ea = packed_profiles_ea + (offset * 8)
    s_len = ida_bytes.get_32bit(str_ea)
    s_str = idc.get_strlit_contents(str_ea + 4)
    return s_str
```

# Findings


- XPC daemons sandboxed on newer devices, unsandboxed on older devices (same iOS version, 12.1.4)
  - Note: older device tested was i5S, newer iXS

- Sandboxed XPC daemons/IOKit drivers may differ among iOS versions
  - And their profile conditions (filters)

- Even more surprising results when you dig deeper

# Attack surface enumeration


- Assumption: we start as a regular app
  - Dev signed or compromised

- Goal: LPE and then kernel code execution
  - LPE: app (mobile) -> root || app -> sandbox escape
  - Direct kernel code execution possible but surface keeps getting smaller/hardened

- Automated; output stored per iOS version and device model
  - So diff among them is possible

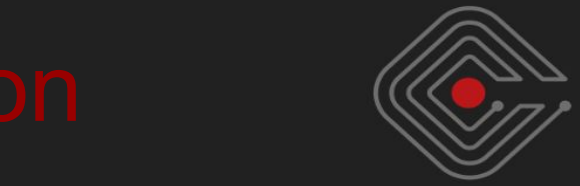
# Attack surface enumeration


- Analyze container.sb and gather all reachable XPC services
  - And the required entitlements (and other conditions)

- Analyze each XPC service's profile and gather all reachable IOKit UserClients

- The diff among iOS releases helps you define attack paths
  - And spend your auditing/reversing time more productively

# container.sb evolution




| iOS version | LOC (SBPL) | Size (bytes) |
|---|---|---|
| 11.2.5 | 3697 | 191469 |
| 11.4 b2 | 4994 | 328012 |
| 12.0 b3 | 8023 | 599395 |
| 12.1 b4 | 8285 | 622517 |
| 12.1 | 8285 | 583150 |
| 12.1.2 | 8285 | 626906 |
| 12.2 b4 | 7845 | 707612 |

# Conclusion


- Common belief that the attack surface is reducing with every iOS release
  - The reality is that it changes
  - May be reducing, may be increasing

- Always double check assumptions/findings at runtime!

- Apple’s platform is a great target for reverse engineering

# References



- Apple's Sandbox Guide, fG!, https://reverse.put.as/wp-content/uploads/2011/09/Apple-Sandbox-Guide-v1.0.pdf
- The Apple Sandbox, Dionysus Blazakis, https://dl.packetstormsecurity.net/papers/general/apple-sandbox.pdf
- TinyScheme, http://tinyscheme.sourceforge.net/
- iOS 8: Containers, Sandboxes and Entitlements, Stefan Esser
- Sandblaster, Razvan Deaconescu
- Hack in the (sand)Box, Jonathan Levin, http://newosxbook.com/files/HITSB.pdf


- **Thanks to co-researchers at CENSUS: Asterios Chouliaras, Alexandros Mitakos**

# Questions

CENSUS
IT Security Works